Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Darko Janković

Razvoj prenosljivih namiznih aplikacij z ogrodjem Qt

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Viljan Mahnič

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Proučite problematiko razvoja prenosljivih namiznih aplikacij za osebne računalnike s poudarkom na možnostih, ki jih v ta namen nudi ogrodje Qt. Predstavite arhitekturo tega ogrodja in njegove najpomembnejše značilnosti. Pridobljena spoznanja uporabite pri razvoju preproste grafične aplikacije - grafičnega vmesnika za program youtube-dl. Razvita aplikacija naj bo prilagojena za izvajanje na treh ciljnih platformah: GNU/Linux, Windows 8 in OS X.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Darko Janković, z vpisno številko **63100176**, sem avtor diplomskega dela z naslovom:

Razvoj prenosljivih namiznih aplikacij z ogrodjem Qt

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Viljana Mahniča,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 9. septembra 2014 Podpis avtorja:

Zahvaljujem se mentorju, izr. prof. dr. Viljanu Mahniču, za strokovno pomoč, nasvete in potrpežljivost pri izdelavi diplomske naloge. Hvala staršem in ostalim sorodnikom za moralno in finančno podporo pri študiju. Hvala tudi vsem ostalim, ki ste mi na kakršenkoli način pomagali vsa ta leta.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
API	Application Programming Interface	aplikacijski programski vmesnik
UML	Unified Modeling Language	splošno namenski modelni jezik
GPL	GNU General Public License	splošno dovoljenje GNU
LGPL	GNU Lesser General Public License	manj splošno dovoljenje GNU
XML	Extensible Markup Language	razširljiv označevalni jezik
DTD	Document Type Definition	definicija tipa dokumenta
SDK	Software Development Kit	komplet razvojnih orodij
IDE	Integrated Development Environment	integrirano razvojno okolje

Povzetek

Namen diplomske naloge je predstaviti problem razvoja prenosljivih aplikacij, ter na primeru prikazati razvoj takšne aplikacije z ogrodjem Qt. V uvodu smo podrobneje opisali namen naloge, v drugem poglavju smo predstavili problem prenosljivih aplikacij. V tretjem poglavju smo natančneje opisali ogrodje Qt, orodja, ki se uporabljajo pri razvoju prenosljivih aplikacij in podrobnosti licenciranja ogrodja Qt. V poglavju, ki sledi, smo opisali aplikacijo, ki je bila razvita, in po korakih opisovali razvoj z orodjem Qt Creator. V petem poglavju smo opisali pripravo okolja za prevajanje izvorne kode na platformah Windows 8, OS X Mavericks in X11 (na Lubuntu GNU/Linux). V šestem poglavju smo preizkusili delovanje na vseh treh platformah, v zaključku pa smo opisali uporabnost rezultatov ter možne izboljšave.

Ključne besede: prenosljive aplikacije, medplatformni razvoj, Qt, namizne aplikacije.

Abstract

The purpose of this thesis is to analyze the problem of developing cross-platform applications and to demonstrate the process of development by creating a simple cross-platform desktop application with Qt framework. In the introduction, the purpose of the thesis was described in detail. In the second chapter, Qt framework, tools for cross-platform development and the details of licensing were presented. In the suceeding chapter there is a description of the application that was developed and a step-by-step process of the development with Qt Creator. In the fifth chapter the process for preparing build environments for Windows 8, OS X Mavericks and X11 (on Lubuntu GNU/Linux) platforms was described. In chapter six the application was tested on all three platforms, and in the last chapter the usability and potential improvements of the results were analyzed.

Keywords: cross-platform applications, cross-platform development, Qt, desktop applications.

Poglavje 1

Uvod

Zaradi množice različnih operacijskih sistemov na področju osebnega računalništva se srečujemo s problemom poganjanja posameznih programov na čim večji množici različnih sistemov. Ena izmed rešitev je, da isti program napišemo večkrat, za vsak sistem posebej. Če malo pomislimo, pridemo do zaključka da je tak pristop slab, saj zahteva veliko časa in v primeru, da gre za komercialno aplikacijo, veliko denarja. Idealno bi bilo, da izvorno kodo napišemo le enkrat in jo prevedemo v izvršljivo aplikacijo za ciljne sisteme. Zaradi relativno dolge zgodovine našega problema imamo v tem trenutku množico rešitev, ki nam omogočajo takšen pristop. V našem primeru smo se odločili za uporabo ogrodja Qt [1]. Prednosti, ki jih ima Qt pred ostalimi ogrodji, so enostavnost razvoja aplikacij, veliko število uporabnikov, odlična dokumentacija, podpora najbolj razširjenim operacijskih sistemom in hitrost aplikacij. V delu bomo prikazali razvoj preproste grafične aplikacije in pripravo okolja za prevajanje izvorne kode za 3 ciljne platforme: Windows 8, OS X in GNU/Linux, ter pri tem podrobneje pregledali grafično orodje Qt Creator. Uporabljene in predstavljene bodo tudi funkcionalnosti, ki omogočajo razvoj bolj zapletenih aplikacij.

Poglavje 2

Prenosljive aplikacije

Prenosljive aplikacije (ang. cross-platform) ali z drugim imenom aplikacije, ki so neodvisne od platforme, v našem primeru operacijskega sistema, delimo na grobo v dve skupini; prve so tiste, ki jih moramo posebej pripraviti za vsako platformo in tiste, ki v enaki obliki delujejo na vseh za katere so namenjene. Prednost drugih je to, da se aplikacija v popolnoma identični obliki poganja na različnih platformah, vendar pa ta pristop zahteva, da so na voljo zunanji programi ki delujejo kot vmesna stopnja v komunikaciji med aplikacijo in platformo. Eden izmed najbolj znanih primerov so aplikacije, napisane v programskem jeziku Java. Ko takšne aplikacije pripravimo za poganjanje, nam prevajalnik pripravi posebno datoteko, ki jo lahko poganjamo samo na računalnikih, na katerih teče javanski navidezni stroj (ang. java virtual machine). Nasprotno nam pa prvi način omogoča, da za poganjanje aplikacij ne potrebujemo nobenih zunanjih programov oz. knjižnic (ang. libraries). Vendar ima zaradi tega razvijalec oz. distributer aplikacije več dela. Za vsako platformo, na kateri želimo poganjati aplikacijo, je potrebno pripraviti okolje za prevajanje kode v izvršljivo datoteko (ang. executable). Namen prenosljivih aplikacij je torej to, da zmanjšamo čas razvoja aplikacij za več platform, uporabnikom pa zagotovimo enotno uporabniško izkušnjo. Slednje pomeni to, da uporabnik ne glede na to, na kateri platformi uporablja aplikacijo, ima le-ta enako strukturo, funkcionalnost in obnašanje. Negativna

posledica tega pa je, da so razvijalci omejeni na skupni imenovalec funkcionalnosti, ki jih ponujajo platforme, za katere razvijajo. Kljub temu, da se uporabi prenosljivo ogrodje za razvoj aplikacij, obstaja možnost, da se bo določena funkcionalnost aplikacije na enem sistemu obnašala drugače kot na ostalih, zaradi različne arhitekture operacijskih sistemov. Tako lahko imamo težave zaradi npr. drugačnega upravljanja s procesi. V izogib težavam te vrste je zato potrebno aplikacijo pred izdajanjem v produkcijo najprej testirati po vnaprej predpisanih scenarijih in popraviti napake ki povzročajo drugačno delovanje. Da bi kar se da zmanjšali verjetnost pojava takšnih težav, je dobro uporabljati preverjena in razširjena orodja, ki so relativno pogosto posodabljana s popravki in izboljšavami.

Poglavje 3

Ogrodje Qt

Ogrodje Qt (kjut) je leta 1995 izdalo podjetje Trolltech, nakar je razvoj prevzela Nokia, potem pa podjetje Digia, vsa tri finska podjetja, od leta 2011 pa za razvoj skrbi Qt Project, katerega pa trenutno vodi že prej omenjena Digia. Ogrodje je v celoti napisano v programskem jeziku C++ in tudi same aplikacije se pišejo v istem jeziku, kar pa ni pravilo, saj nam ogrodje omogoča pisanje kode tudi v drugih popularnih jezikih. Za ogrodje Qt kot primerno ogrodje za razvoj prenosljivih aplikacij smo se odločili zato ker deluje na širokem naboru različnih platform, omogoča svobodno izbiro pri odločanju glede licenciranja, je prva izbira razvojnega orodja mnogih podjetij in projektov kot so Sennheiser, Panasonic, KDE, VLC, Mathematica, ...[1] in nenazadnje, omogoča hitro vzpostavitev razvojnega okolja in nudi izjemno dobro integrirano razvojno okolje (Qt Creator), ki zagotavlja čim krajše cikle od načrtovanja do postavitve (ang. deployment) aplikacije. Podprte platforme za razvoj namiznih aplikacij so Linux 23 in 64-bit, Mac, Windows 32 in 64-bit. Za okolje *Visual Studio* za razvoj aplikacij na platformi Windows, obstaja dodatek (Add-in), ki zagotovalja podporo pri razvoju z ogrodjem Qt, v kolikor razvijalec ne želi uporabljati priloženega razvojnega okolja.

[1] http://qt.digia.com/Qt-in-Use/

3.1 Arhitektura

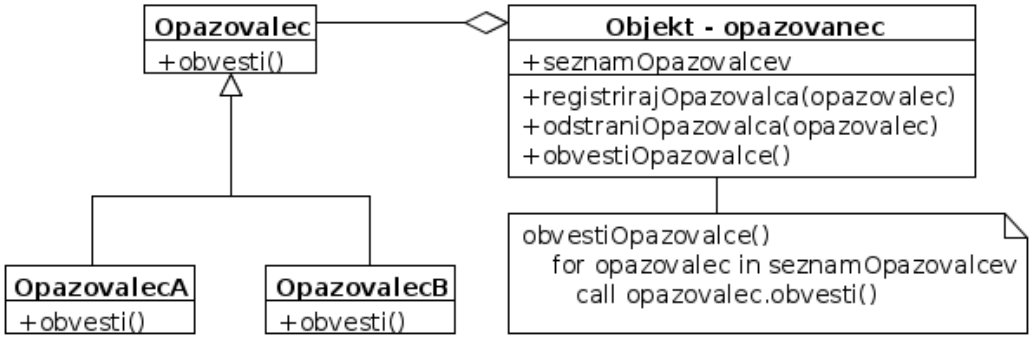
3.1.1 Domorodni izgled

Zelo pomemben aspekt prenosljivih aplikacij je domorodni (ang. native) izgled. Qt ima rešeno to z uporabo domorodnih aplikacijskih programskih vmesnikov (ang. application programming interface - API), tako da se uporabljajo barve, oblike, slogi pisave, metrike, ... platforme, na kateri teče aplikacija. Sprva je Qt uporabljal lasten sistem za emulacijo ciljne platforme, vendar se je sčasoma to izkazalo za slabšo alternativo. Novejše verzije uporabljajo domorodne klice za izrisovanje.

3.1.2 Signali in reže

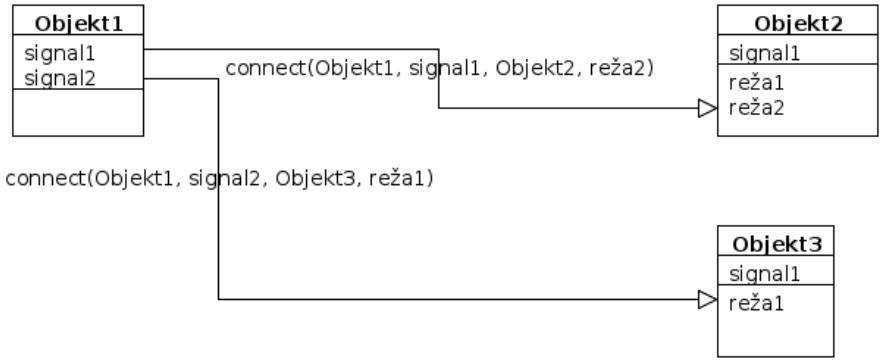
Signali in reže (ang. signals and slots) so uporabna funkcionalnost ogrodja Qt za komunikacijo med objekti. Z njimi lahko preprosto implementiramo vzorec *opazovalca*. Primerni so za asinhrono obveščanje o dogodkih, kot je npr. klik na gumb. Vzorec opazovalca je eden izmed vzorcev načrtovanja programske opreme (ang. software design pattern), ki se največ uporablja za implementacijo obveščanja o dogodkih med objekti. Vzorec sloni na principu, da obstaja nek objekt (opazovanec) s seznamom opazovalcev, ki so obveščani o spremembah stanja objekta. Obveščanje ponavadi poteka tako, da objekt kliče metodo, katero implementira opazovalec. Na sliki 3.1 je prikazan diagram UML (Unified Modeling Language) vzorca opazovalca. *Opazovanec* ima seznam opazovalcev, ki jih obvešča o dogodkih, metodi za registracijo in odstranjevanje opazovalca ter metodo za obveščanje vseh opazovalcev. Vmesnik *Opazovalec* definira metodo *obvesti()*, ki jo implementirata opazovalca *OpazovalecA* in *OpazovalecB*. Relacija med vmesnikom in opazovancem, ki se imenuje *agregacija*, pomeni da ima lahko opazovanec več registriranih opazovalcev, nasprotno pa opazovalec samo en objekt, ki ga opazuje. Hkrati agregacija zahteva da sta v relacijo vpletena največ dva razreda in da življenjski cikel opazovalca ni odvisen od cikla opazovanca; z drugimi besedami, uničenje

objekta ki je opazovan, ne vpliva na njegove opazovalce [12].



Slika 3.1: Vzorec opazovalca - UML diagram

Sistem signal/reža je osrednji del ogrodja Qt, ki olajša razvoj grafičnih vmesnikov. Signal se sproži ob nekem dogodku in Qt ima množico predefiniranih signalov, vedno pa lahko po potrebi dodamo lastne signale. Reža je metoda, ki se pokliče kot odziv na sprejeti signal. Tudi reže so v velikem številu že predefinirane, zaradi specifičnih zahtev glede delovanja aplikacij pa se največkrat implementirajo lastne reže. Na sliki 3.2 je prikazan primer vezave signalov na reže.



Slika 3.2: Sistem signal/reža - diagram

Sistem zagotavlja identične tipe parametrov (ang. type safety) tako, da zahteva enak podpis (ang. signature) signala in reže. To ne drži v primeru, ko ima reža definiranih manj argumentov kot jih ima signal; obstaja tudi možnost, da ne sprejema nobenega argumenta. V splošnem ni omejitve pri številu in tipih argumentov, ki jih dodajamo, dokler je upoštevano prejšnje pravilo. Razred, ki oddaja signale ne ve nič o režah, ki signale sprejemajo, zato pravimo da je sistem *šibko sklopljen* (ang. loosely coupled).

Vsi razreda tipa *QObject* in razredi, ki slednjega razširjajo, lahko uporabljajo signale in reže. QObject je osnovni razred in ga razširjajo vsi ostali razredi ogrodja Qt.

Primarna naloga rež je odzivanje na sprejete signale, vendar se jih lahko uporablja tudi kot običajne metode. Razredi nimajo informacij o tem, če oddajajo signale in enako reže ne vedo, če so kakšni signali povezani na njih. Na posamezno režo lahko vežemo neomejeno število signalov, in prav tako lahko en signal vežemo na poljubno število rež. Obstaja tudi možnost vezave signala na signal, s čimer dobimo verige signalov. S tem dosežemo proženje drugega signala po tem, ko se je sprožil prvi [13].

Signali

Ko se signal sproži, se povezane reže takoj izvršijo. Delovanje sistema je neodvisno od operacij, ki se izvajajo na grafičnem vmesniku. Če je vezanih več rež na isti signal, se najprej pridobijo vse reže, ki se nato izvajajo po istem vrstnem redu, kot so bile povezane.

Reže

Reže se od navadnih metod razlikujejo le po tem, da jih lahko povežemo s signali. Signali in reže so počasnejši sistem od alternativnega sistema povratnih klicev (ang. callbacks), ki ga uporabljajo nekatera konkurenčna ogrodja. V povprečju je pošiljanje signala desetkrat počasnejše kot neposredno klicanje prejemnika, zaradi lociranja prejemnika, varnega iteriranja čez vse povezave med signali in režami (preverjanje, če je bil prejemnik uničen) in serializa-

cije parametrov. Fleksibilnost in preprostost v glavnem odtehtata počasnejše delovanje, ki je za človeka praktično nezaznavno.

Listing 3.1: Primer vezave signala na režo

```
QObject::connect(info, SIGNAL(finished(int)), this,
SLOT(getInfo()));
```

Koda 3.1 prikazuje uporabo sistema signal/reža. Objekt *info* tipa *QProcess* smo s signalom *finished* vezali na metodo *getInfo()*, ki se nahaja v trenutnem objektu *this*. S tem smo dosegli da se metoda *getInfo()* začne izvajati v tistem trenutku, ko se proces predstavljen z objektom *info* konča.

3.1.3 Metaobjektni prevajalnik

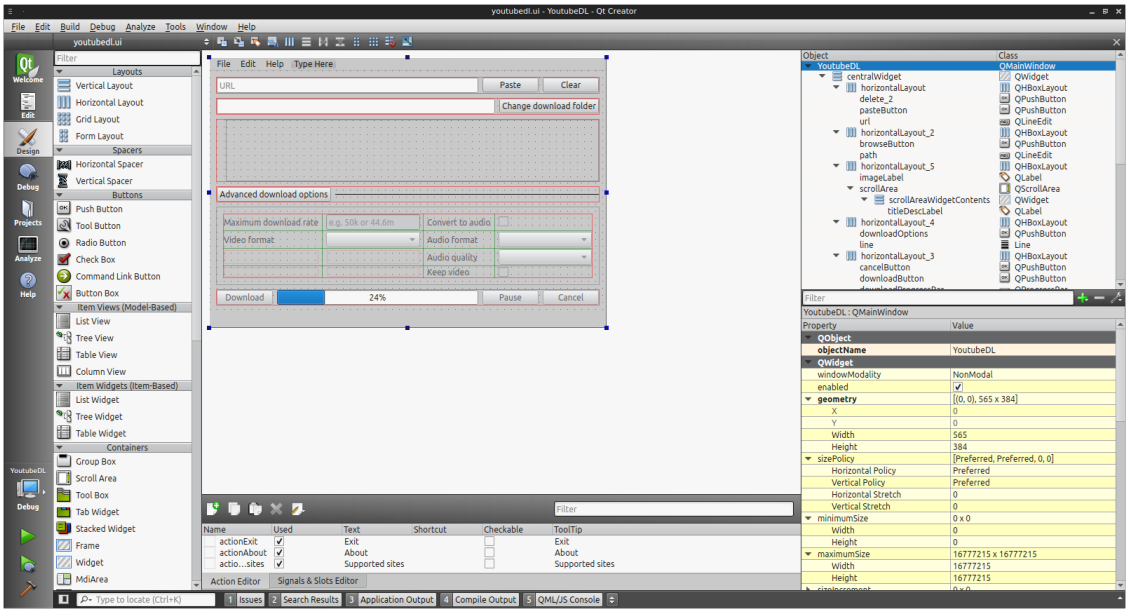
Metaobjektni prevajalnik (ang. metaobject compiler - moc) se kliče pri prevajanju aplikacij, napisanih z ogrodjem Qt, in sicer interpretira specifične Qt makroje v kodo C++ z metapodatki o razredih, ki jih aplikacija uporablja. S tem Qt razširja možnost C++, tako da lahko uporabljamo napredne funkcionalnosti kot je že prej omenjen sistem signali/reže.

3.1.4 Moduli

Qt je razdeljen v več modulov ki zagotavljajo množico funkcionalnosti. Modul *Core* vsebuje osnovne funkcije ki jih potrebuje vsak Qt projekt, *Gui* zagotavlja grafične elemente aplikacije, *Multimedia* omogoča funkcionalnosti, vezane na zvok, slike in multimedijske naprave, kot je kamera. Na naslovu `https://qt-project.org/doc/qt-5/qtmodules.html` so našteti in opisani vsi moduli.

Listing 3.2: Dodajanje potrebnih modulov v projektno datoteko

```
QT += core gui network
```



Slika 3.3: Qt Creator - način Design

3.2 Qt Creator

Qt Creator je integrirano razvojno okolje (ang. integrated development environment - IDE) za razvijalce, ki ustvarjajo aplikacije za različne namizne in mobilne platforme. Namen Qt Creatorja je omogočiti razvijalcem lažjo uporabo samega ogrodja Qt in s tem hitrejše načrtovanje in razvoj aplikacij ter uporabniških vmesnikov. Vsebuje orodja, ki nam pomagajo skozi celoten postopek razvoja, od ustvarjanja projekta do prevajanja kode za ciljne platforme. Za vsak aspekt razvoja aplikacije je v Qt Creatorju na voljo poseben zavihek oz. način (ang. mode). V zavihku *Welcome* je prikazan seznam nazadnje odprtih projektov in povezave do primerov in vodičev. V načinu *Edit* se pri razvoju verjetno preživi največ časa, saj se v tem načinu piše programska logika aplikacije. Način *Design* omogoča preprosto urejanje grafičnega vmesnika z dodajanjem elementov po principu povleci in spusti (ang. drag and drop). Na sliki 3.3, kjer je prikazan način Design, vidimo množico razpoložljivih grafičnih gradnikov, ki jih lahko dodajamo, urejamo in odstra-

njujemo. Vsak gradnik ima atribute, ki jih lahko spreminjamo in v realnem času opazujemo spremembe. V tem načinu lahko dodajamo predefinirane signale in reže na standardne gradnike, kot je npr. gumb za izhod iz programa. Namen načina *Debug* je lociranje napak v logiki aplikacij in analiza delovanja same aplikacije. V zavihku *Projects* je seznam vseh obstoječih projektov in vmesnik za specificiranje ukazov za grajenje aplikacije. Spreminjamo lahko parametre ukazov qmake in make, v primeru da nam privzeti parametri, ki se generirajo ob kreiranju novega projekta, ne ustrezajo. Poleg tega so v tem zavihku nastavitve za urejevalnik kode, kjer lahko spreminjamo nastavitve kot npr. dolžina tabulatorja, poravnava večvrstičnih ukazov in pravila za pozicioniranje zavitih oklepajev. Način *Analyze* je primeren predvsem pri razvoju aplikacij za naprave z omejeno velikostjo glavnega pomnilnika. V tem načinu lahko natančno analiziramo notranje delovanje (signali in reže, izrisovanje) mobilnih aplikacij. V zavihku *Help* najdemo navodila in pomoč za razvoj aplikacij.

3.2.1 Razhroščevalnik

Ang. debugger, nam omogoča analizo napak v programu. *Qt Creator* vsebuje vtičnik ki služi kot vmesnik za dostopanje do zunanjih razhroščevalnikov kot so GDB, LLDB, CDB [7]. Omogoča nam prekinjanje izvajanja programa, premikanje skozi program po posameznih vrsticah oz. ukazih, nastavljanje prekinitvenih točk (ang. breakpoints), analizo programskega sklada in lokalnih ter globalnih spremenljivk. Ker omogoča razhroščevanje programov, napisanih v jeziku C++, nismo omejeni samo na programe, napisane z ogrodjem Qt, ampak ga lahko uporabljamo tudi za ostale projekte. Pri razvoju iger za platformo GNU/Linux se npr. uporablja v podjetju Steam [5].

3.3 qmake

Qmake je orodje za avtomatizacijo procesa prevajanja in je sestavni del ogrodja Qt, vendar je kljub temu uporabno tudi za aplikacije napisane v

drugih programskih jezikih. Naloga qmake je, da generira *Makefile* glede na informacije v projektni datoteki (.pro), ki jo ustvari razvijalec. *Makefile* datoteka je konfiguracijska datoteka za program *make*. Sestavljena je iz množice pravil, ki določajo, kateri prevajalniki se naj uporabijo, navodila za povezovalnik ipd. Pisanje takšnih konfiguracijskih datotek je lahko časovno potratno, zato ponavadi uporabljamo orodja za avtomatizacijo, kot je qmake.

3.3.1 Make in makefile

Make je orodje za samodejno grajenje (ang. build) izvršljivih programov iz izvorne kode s pomočjo konfiguracijskih datotek makefile. Uporabno je tudi za upravljanje s projekti drugačnih tipov, kjer imamo izvorne datoteke, in želimo ob njihovi spremembi posodobiti izhodne datoteke. Make je leta 1976 ustvaril Stuart Feldman, ki je v tistem času bil zaposlen v podjetju Bell Labs. Leta 2003 je kot avtor tega orodja dobil nagrado ACM (Association for Computing Machinery) [14] Software System Award.
GNU Make je standardna implementacija za platforme GNU/Linux in OS X. Osnovno funkcionalnost razširja s funkcijami, ki nadomeščajo uporabo skriptnega jezika v datoteka makefile. Uporablja se za grajenje zbirke prevajalnikov *gcc* [15] in jedra *Linux* [16].
Nmake je implementacija podjetja Microsoft, ki pride v paketu s programom Visual Studio in se uporablja na platformi Windows [17].
Make se tradicionalno poganja iz ukazne vrstice, in če mu ne podamo argumentov, sledi navodilom prve tarče (ang. target) v datoteki makefile, ki jo bomo podrobneje pogledali v nadaljevanju. Make preveri čase zadnjih sprememb izvornih datotek in cilja, in če se je ena od izvornih datotek spremenila odkar je bila tarča zgrajena, se požene ustrezen ukaz, definiran v makefile.
Ko se program make požene, najprej v trenutnem direktoriju išče za datotekami makefile, ki so ponavadi poimenovane *Makefile* oz. *makefile*, in sledi navodilom za grajenje specificirane tarče. Jezik, uporabljen v datotekah makefile, ima podobnosti z deklarativnimi programskimi jeziki, kot je npr. jezik *SQL* [18]. V primeru samodejnega grajenja programov za različne platforme

je potrebno določevati različne prevajalnike, ki se bodo uporabljali, za kar pa make ni najbolj primerno orodje. Ta problem nam rešuje prej omenjeni qmake. Datoteka makefile je sestavljena iz množice pravil (ang. rule), ki definirajo tarče. Tarče so definirane z imenom, seznamom vhodnih datotek in ukazi.

Listing 3.3: Primer tarče v datoteki makefile

```
main.o: main.cpp youtubedl.h
        $(CXX) -c $(CXXFLAGS) $(INCPATH) -o main.o main.cpp
```

V kodi 3.3 je ime tarče *main.o*, izvorni datoteki sta *main.cpp* in *youtubedl.h*, ukaz pa je podan v drugi vrstici. Izrazi, ki se začnejo z znakom *$* in so oviti v oklepaje, so spremenljivke, ki se jih ponavadi definira na začetku datoteke. To so ponavadi prevajalniki, posebni argumenti in daljši ukazi, ki se uporabljajo v procesu grajenja [19].

3.4 Licenciranje

Qt nam omogoča tri tipe licenc za aplikacije, ki jih ustvarimo.

Komercialna licenca je primerna za razvoj aplikacij, katerih izvora ne želimo distribuirati, oz. se ne strinjamo s pogoji svobodnih licenc, kot je GPL. S pridobitvijo komercialne licence dobimo tudi naprednejšo verzijo ogrodja Qt, primerno za večja podjetja ki razvijajo kompleksno programsko opremo.

LGPL 2.1 [9] nam omogoča licenciranje aplikacij pod le-to licenco, ali pa našo lastniško, pod pogojem, da uporabljene knjižnice ogrodja Qt ne predstavljajo več kot 5% naše aplikacije in še nekaterimi ostalimi pogoji [8].

GPL 3.0 [10] bomo uporabili v primeru, da v našo aplikacijo vključujemo programe ki imajo enako licenco, ali pa sami želimo izdati aplikacijo pod to licenco.

Ob izbiri licence moramo biti pozorni na posledice, ki jih le-ta prinese. Največ svobode razvijalcu omogoča komercialna licenca; sami se odločimo pod kakšnimi pogoji in na kakšen način bo naša aplikacija distribuirana. Licenci LGPL in GPL pa sta bolj svobodni z vidika uporabnika, saj ima le-ta v določenih situacijah pravico zahtevati izvorno kodo aplikacije. LGPL omogoča da izvorne kode ni potrebno zagotoviti uporabniku, v primeru da je aplikacija prevedena z dinamičnim povezovanjem. Če se odločimo distribuirati statično prevedeno aplikacijo, ki vsebuje vse potrebne knjižnice ogrodja Qt, moramo na zahtevo uporabnika zagotoviti izvorno kodo pod LGPL oz. GPL licenco.

Podrobnosti o licencah za posamezne module, ki sestavljajo ogrodje Qt, so dostopne na `http://qt-project.org/doc/qt-5/licensing.html`.

3.4.1 Dinamično povezovanje

Ko se odločamo kateri pristop vključevanja zunanjih knjižnic bomo uporabili pri prevajanju, imamo ponavadi pred seboj dve izbiri. Dinamično povezovanje nam omogoča, da program pri izvajanju poišče knjižnice, potrebne za delovanje, na sistemu kjer se poganja. Zaradi tega je prevedena izvršljiva datoteka majhna, vendar lahko pride do problema, da nekaterih knjižnic na gostujočem sistemu ni, tako da jih mora uporabnik namestiti, saj v nasprotnem primeru program ne bo deloval. Dinamično povezovanje lahko zmanjša porabo sistemskih virov (zasedenost diska, glavnega pomnilnika in predpomnilnika), ker zaradi uporabe deljenih knjižnic (ang. shared libraries) več programov uporablja isto knjižnico, in je zato potrebna samo ena instanca knjižnice v glavnem pomnilniku. Potencialna prednost dinamičnega povezovanja je posodabljanje zunanjih knjižic (v našem primeru ogrodja Qt), saj uporabniku ni potrebno dostaviti nove verzije aplikacije samo zaradi posodobitve zunanjih knjižnic. Seveda se to lahko izkaže kot problem v primeru, da zunanje knjižnice ne ohranjajo združljivosti s starimi verzijami (ang. backward compatibility).

3.4.2 Statično povezovanje

Pri tem pristopu se pri prevajanju v izvršljivo datoteko vključijo potrebne knjižnice, zato je le-ta večja v primerjavi s tisto, ki nastane pri dinamičnem povezovanju, ampak rešimo problem manjkajočih knjižnic. Posledica, ki jo potegne za seboj statično povezovanje je ta, da moramo skupaj z izvršljivo datoteko uporabniku dati na voljo tudi izvorno kodo skladno z licenco ogrodja Qt v primeru, da smo razvijali z odprtokodno različico Qt. Statično povezovanje olajša distribucijo aplikacije, saj končni uporabnik preprosto požene izvršljivo datoteko brez potrebe po nalaganju zunanjih programov na sistem. Statično povezovanje lahko pohitri začetni zagon aplikacije, saj se v glavni pomnilnik naložijo vse potrebne knjižnice hkrati, in jih ni potrebno nalagati posebej.

Poglavje 4

Razvoj aplikacije

4.1 Razvojno okolje

Aplikacijo smo razvili s pomočjo programa Qt Creator 3 z ogrodjem Qt verzije 5.3.1 na GNU/Linux platformi.

4.2 Opis aplikacije

Razvita je bila grafična aplikacija, ki omogoča pridobivanje informacij o multimedijskih virih (zvok in video), prenos vsebin in pretvarjanje formata prenesenih vsebin. Aplikacija je grafični vmesnik za program youtube-dl [21], ki je načrtovan kot konzolna aplikacija. Program youtube-dl je napisan v jeziku Python [20], ki je že v osnovi načrtovan kot prenosljiv jezik. Youtube-dl omogoča prenašanje virov z množice spletnih strani, ki ponujajo zvočne in video vsebine. Je brezplačno orodje s svobodno licenco in javno objavljeno kodo, ki jo lahko kdorkoli uporablja in spreminja po želji. Namen razvite aplikacije je poenostaviti uporabniku uporabo programa youtube-dl, to je pa doseženo z grafičnim vmesnikom ki omogoča naslednje funkcije:

- Pridobivanje naslova, opisa in slike multimedijskega vira, dosegljivega na spletu
- Prenos vira na računalnik

- Nastavljanje največje hitrosti prenosa
- Pretvarjanje vira v zvok različnih formatov
- Nastavljanje kvalitete pretvorjene datoteke
- Izbira za izbris oz. ohranitev originalne datoteke

Aplikacija je bila razvita za tri najbolj razširjene namizne platforme: Windows, OS X in GNU/Linux. Zagotovoljen je domoroden izgled na platformi kjer aplikacija teče, integracija s specifičnimi nastavitvami sistema (barve, pisave) in enaka uporabniška izkušnja na vseh platformah. Za popolno funkcionalnost so potrebni program youtube-dl ter knjižnice za pretvarjanje med multimedijskimi formati, katerih namestitev se razlikuje med platformami in je v nadaljevanju tudi opisana.

4.3 Grafični vmesnik

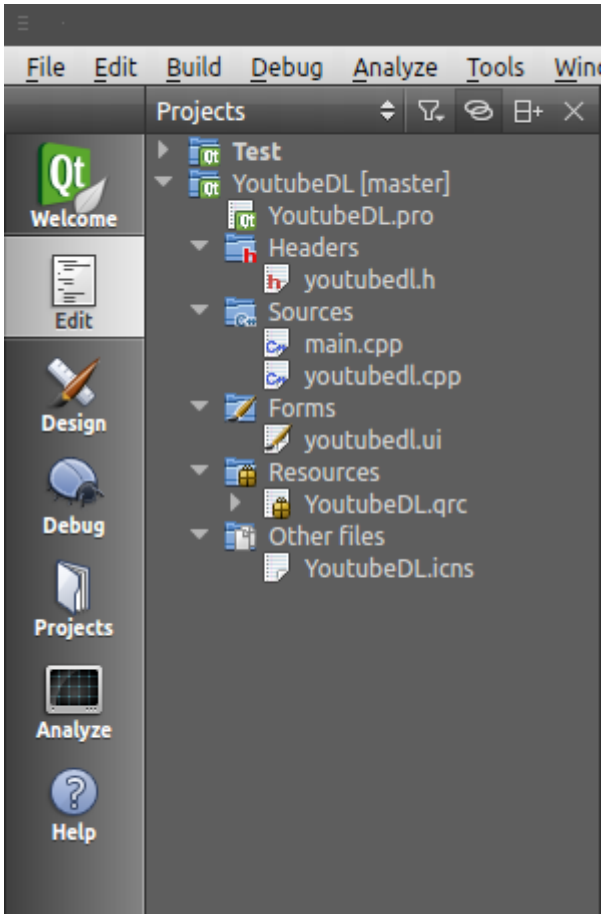
Qt Creator nam omogoča samodejno ustvarjanje ogrodja za grafični vmesnik. V čarovniku za novo aplikacijo smo izbrali projekt tipa *Qt Widgets Application*, ki pripravi osnovno strukturo namizne aplikacije.

Na sliki 4.1 je prikazana struktura projekta kot nam jo prikaže Qt Creator. Vse datoteke so v enem direktoriju, vendar jih Qt Creator glede na vsebino razdeli po kategorijah v mape.

YoutubeDL.pro je projektna datoteka, ki vsebuje navodila za program qmake ki zgradi (ang. build) aplikacijo.

Listing 4.1: Projektna datoteka

```
QT       += core gui network
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
TARGET = YoutubeDL
TEMPLATE = app
SOURCES += main.cpp\
        youtubedl.cpp
HEADERS  += youtubedl.h
FORMS    += youtubedl.ui
```



Slika 4.1: Struktura projekta v Qt Creatorju

Headers mapa vsebuje zaglavne datoteke, v katerih so vnaprejšnje deklaracije razredov, funkcij, spremenljivk itd., kot pri navadnih C in C++ projektih.

Sources vsebuje izvorne datoteke, v katerih se nahaja glavna logika aplikacije kot npr. vezava signalov na reže in implementacija slednjih. Qt Creator nam samodejno generira datoteko *main.cpp*, kjer se nahaja funkcija *main*, ki poskrbi za ustvarjanje in prikazovanje aplikacije ob zagonu.

Forms združuje datoteke s končnico *.ui*, ki so pravzaprav *xml* datoteke, v katerih določamo izgled aplikacije. Qt Creator nam ne omogoča neposrednega spreminjanja vsebine teh datotek, ampak lahko z njimi manipuliramo le preko načina *Design*. V tem načinu lahko preprosto dodajamo nove elemente in jih razporejamo ter dodajamo signale, kar nam omogoča zelo hiter razvoj vmesnika. V predogledu pa je prikazan vmesnik, kot bo viden na platformi kjer trenutno razvijamo.

Resources vsebuje datoteke s končnico *qrc* tipa *Qt Resource*, ki temelji na standardu XML. Sistem Qt Resource je prenosljiv način vgrajevanja binarnih datotek v izvršljivo datoteko aplikacije. Primeren je za dodajanje virov, ki so za aplikacijo vedno potrebni (slike, prevodi, ...). Vire lahko preprosto dodajamo v načinu Edit.

Listing 4.2: Primer datoteke qrc

```
<RCC>
    <qresource prefix="/">
        <file>YoutubeDL.png</file>
    </qresource>
</RCC>
```

Ko imamo vir dodan, do njega preprosto dostopamo s predpono *prefix* in imenom datoteke *file*.

Listing 4.3: Dostopanje do virov

```
QApplication a(argc, argv);
```

```
a.setWindowIcon(QIcon(":/YoutubeDL.png"));
```

Other files vsebuje vse ostale datoteke ki ne spadajo v prej naštete kategorije.

4.3.1 Grajenje grafičnega vmesnika

Ob kreiranju novega projekta se ustvari ogrodje grafične aplikacije, ki je sestavljeno iz glavnega okna tipa *QMainWindow*, centralnega vsebovalnika tipa *QWidget*, vrstice z menijem tipa *QMenuBar*, statusne vrstice *QStatusBar* in orodne vrstice, ki je objekt tipa *QToolBar*. Vsi gradniki so gnezdeni v glavno okno in razen menija, v centralni vsebovalnik. Za urejanje vmesnika se uporablja način Design.

Za dodajanje elementov v meni aplikacije preprosto vstavljamo nize, ki se samodejno pretvorijo v gradnike, le-tem pa lahko dodajamo še gnezdene elemente, ki se prikažejo ob kliku. Dodali smo tri standardne elemente v meni; to so *Datoteka*, *Uredi* in *Pomoč*. Element Datoteka vsebuje akcijo *Izhod* za izhod iz aplikacije, Uredi vsebuje akciji *Prilepi* in *Počisti* za lepljenje niza iz odložišča oz. za nastavljanje vmesnika na začetne vrednosti. Vnos Pomoč vsebuje akcijo *O programu* ter *Podprte strani*, ki odpreta okno z informacijami. Na akcijo Izhod smo dodali predefinirani signal *triggered()*, ki se pošlje ob kliku na akcijo, in režo *close()* ki zapre aplikacijo. Kot prejemnika (ang. receiver) smo izbrali v spustnem meniju našo aplikacijo. Akcijama Prilepi in Počisti smo prav tako dodali signal triggered() in predefinirani reži *paste()* oz. *clear()*, ter kot prejemnika določili vnosno polje(ki ga dodamo prehodno) za naslov vira, ki po akciji Prilepi dobi vsebino odložišča, po akciji Počisti pa se vsebina izbriše.

Za razporeditev gradnikov na vmesniku in določanje relativnih razdalj med posameznimi gradniki smo uporabili vsebovalnike *QHBoxLayout*, ki razporedijo vgnezdene gradnike horizontalno. Da lahko določimo prejemnika signala v spustnem seznamu, je potrebno prejemnika predhodno ustvariti. Za vnosno polje, ki sprejme naslov vira, smo dodali gradnik tipa *QLineEdit*.

Poleg akcij v meniju smo dodali še dva gumba tipa *QPushButton*, ki imata enaki funkciji kot akciji Prilepi in Počisti v meniju.

V drugi skupini gradnikov se nahaja gradnik QLabel, ki uporabniku prikazuje informacijo o mapi kamor se bo vir prenesel, poleg njega pa je gumb, ki odpre dialog za izbiro ciljne mape.

Za osrednji del, kjer se bodo prikazovali slika, naslov in opis vira, smo dodali še en vsebovalnik QHBoxLayout. Vanj smo dodali gradnik *QLabel*, ki je odgovoren za prikaz slike, ter poleg njega še en vsebovalnik tipa *QScrollArea*. V slednjega smo dodali še en gradnik QLabel, ki skrbi za prikaz naslova in opisa vira. V gradnik QScrollArea smo ga vgnezdili zato, ker je besedilo opisa spremenljive dolžine, QScrollArea pa ob daljših besedilih doda ob straneh drsnika za premikanje v horizontalni in vertikalni smeri, kar omogoči uporabniku pregled nad celotno vsebino, brez da bi se spreminjale dimenzije celotnega vmesnika.

Za naslednjo skupino gradnikov smo ponovno dodali vsebovalnik QHBoxLayout, vanj pa dodali gumb ter črto tipa *Line*, ki vizualno ločuje različne sklope aplikacije. Gumb skrbi za prikaz privzeto skritega vmesnika za nastavljanje naprednih možnosti. Za odzivanje na klike smo implementirali lastno režo, ki ob klicu prikaže skriti vmesnik. Za implementacijo lastne reže smo v zaglavno datoteko v sekcijo *private slots* dodali deklaracijo metode, v izvorni datoteki pa jo implementirali.

Vmesnik z naprednimi možnostmi smo naredili s pomočjo vsebovalnika *QGridLayout*, ki ustvari mrežo s polji, v katera vstavljamo željene gradnike. Dodali smo vnosno polje, v katerega uporabnik vpiše največjo dovoljeno hitrost prenosa, potrditveni polji tipa *QCheckBox*, s katerima uporabnik označi če želi pretvoriti format prenesene datoteke, in če želi da se po pretvorbi originalna datoteka ne izbriše. Poleg tega smo dodali še dva spustna seznama QComboBox, v katerih uporabnik določa v kateri avdio format se naj pretvori prenesena datoteka in kvaliteto zvoka pretvorjene datoteke. Vsak omenjeni gradnik dopoljnjujejo gradniki QLabel, ki uporabniku opisujejo namen vsakega gradnika.

Zadnja skupina gradnikov je kot vse prejšnje, v lastnem vsebovalniku QHBoxLayout, in sestoji iz 4 gradnikov. Najpomembnejši je gumb *Prenos*, ki ob kliku preveri podani spletni vir in če ni napak, v ozadju pokliče program youtube-dl, ki opravi prenos. Stanje prenosa v procentih prikazuje indikator napredka tipa *QProgressBar*, tako da ima uporabnik informacijo o procesu prenašanja datoteke. Poleg je gumb, ki omogoča pavzo in nadaljevanje prenosa, ter gumb za preklic operacije.

4.3.2 Implementacija programske logike

V kolikor ne uporabljamo predefiniranih signalov in rež, ter želimo naprednejše odzivanje na uporabnikovo interakcijo z vmesnikom, je potrebno dodati lastno kodo, ki izboljša uporabniško izkušnjo. V nadaljevanju bomo opisali logiko, ki skrbi za odzive na uporabnikove akcije.

Med akcijami v menijih, ki ne uporabljajo predefiniranih rež sta *O programu* in *Podprte strani*. Ob kliku na prvo akcijo se uporabniku prikaže manjše okno z imenom, ikono in opisom aplikacije. To smo storili tako, da smo implementirali lastno režo, ki se odziva na signal *triggered()* in je vezana na to akcijo.

Listing 4.4: Reža za odzivanje klika na akcijo O programu

```
void YoutubeDL::on_actionAbout_triggered()
{
    QMessageBox::about(this, tr("About YouTubeDL"),
    tr("This is a GUI for youtube-dl.<br/>
    <br/>Author: Darko Jankovic"));
}
```

V kodi 4.4 je prikazana implementacija reže, ki se izvrši ob kliku na akcijo O programu. Qt ima pripravljen razred *QMessageBox* za standardna sporočila uporabniku, ki se prikazujejo v posebnem oknu. Tipi sporočil so sledeči:

about - prikaz okna z osnovnimi informaciji o aplikaciji

aboutQt - prikaz okna z osnovnimi informacijami o ogrodju Qt

critical - obvestilo o izredni situaciji

information - okno za informiranje uporabnika

question - potrditveno okno

warning - okno za opozarjanje uporabnika

Ker želimo, da okno prikaže uporabniku osnovne informacije o aplikaciji, smo uporabili tip *about*. Ta tip samodejno poišče priloženo ikono in jo prikaže poleg naslova, ki je drugi argument, in opisa, ki je tretji argument. Prvi argument je referenca na starša tega okna, ki je v našem primeru *this* - referenca na trenutni kontekst. Podobno smo za akcijo *Podprte strani* uporabili isti razred, ampak tipa *information*, kot opis pa prikazali seznam vseh strani, s katerih lahko pridobivamo informacije o multimedijskih virih.

Prva skupina gradnikov sestoji iz vnosnega polja za spletni vir, gumb za lepljenje iz odložišča in gumb za ponastavljanje vmesnika. Željeno obnašanje aplikacije je, da se ob spremembi besedila v polju za vir, pokliče metoda ki bo poskusila dobiti informacije o viru in jih prikazati. To smo storili z implementacijo reže za signal *textChanged()*, ki se sproži ob spremembi besedila vnosnega polja.

Listing 4.5: Reža za odzivanje na spremembo besedila v vnosnem polju

```
void YoutubeDL::on_url_textChanged(const QString &arg1) {
    // Check if URL is correct and valid
    if (arg1.isEmpty() || !QUrl(arg1).isValid()) {
        if (NULL != info && info->state() != 0)
            info->kill();
        this->resetInterface();
    } else {
        ui->pasteButton->setDisabled(true);
        // Get video info
        QString program = "youtube-dl";
        QStringList arguments;
        arguments << "--get-thumbnail" << "--get-title" <<
```

```
            "--get-description" << "--no-playlist" << arg1;
            info = new QProcess(this);
            info->start(program, arguments);

            QObject::connect(info, SIGNAL(readyReadStandardError()),
            this, SLOT(printError()));
            QObject::connect(info, SIGNAL(finished(int)),
            this, SLOT(getInfo()));

            QMovie *movie = new QMovie(":images/loader.gif");
            ui->titleDescLabel->setAlignment(Qt::AlignCenter);
            ui->titleDescLabel->setMovie(movie);
            movie->start();

    }
}
```

Reža dobi kot argument novo besedilo in najprej preverimo, če je argument veljaven spletni naslov, nato zaradi specifičnih robnih primerov preverimo če že obstaja proces, ki pridobiva informacije o viru, in če res obstaja, ga ubijemo in na koncu postavimo vmesnik na prvotno stanje (slika, naslov in opis vira). Osrednjo vlogo ima objekt *info* tipa *QProcess*. Razred QProcess se uporablja za klicanje zunanjih programov in komuniciranje z le-temi. Izhod (ang. output) procesov lahko spremljamo preko dveh predefiniranih kanalov: kanal za standardni izhod (ang. standard output) in kanal za napake (ang. standard error). QProcess odda signal *readyRead()*, ko so podatki na voljo na trenutnem kanalu. Podobno lahko preko standardnega vhoda pošiljamo procesu podatke. Poizvedujemo lahko po različnih lastnostih procesa kot je trenutno stanje (ne teče, se zaganja, teče), izhodni kanal na kateremu prestrezamo izhod in informacija o napaki (npr. napaka pri zaganjanju, zrušitev, ipd.). Za pridobivanje informacij o procesu imamo na voljo sinhroni API, ki je primeren za konzolne aplikacije, in komunikacijo preko signalov in rež, ki za razliko od sinhronega načina ne blokira trenutne niti medtem ko se čaka na signal. Za grafične aplikacije je torej primernejša komunikacija s signali, v nadaljevanju pa so našteti signali, ki smo jih uporabili.

readyReadStandardOutput se sproži, ko so na voljo podatki na standardnem izhodu procesa

readyReadStandardError signal se se sproži, ko so na voljo podatki na kanalu za napake

finished signal se pošlje, ko se proces zaključi

Signal *readyReadStandardOutput* smo vezali na režo *printError()*, ki uporabnika opozori da je prišlo do napake. Signal *finished(int)* smo povezali na režo *getInfo()*, ki se pokliče ob zaključku procesa in na vmesniku prikaže sliko, naslov in opis vira, ki ga je uporabnik podal. Informacije o viru dobimo preko metode *readAllStandardOutput()*, ki vrne vse podatke na standardnem izhodu v obliki objekta tipa *QString*. Iz tega objekta se preberejo podatki s katerimi naprej manipuliramo.

Drugo skupino gradnikov sestavljata element, ki prikazuje pot do mape, kamor se bo shranil preneseni vir in gumb za izbiro mape. Ob kliku na gumb se pokliče reža, ki uporabniku pokaže dialog za izbiro mape. Qt ima za ta namen implementiran razred *QFileDialog*. Prikaz datotek in map v dialogu lahko filtriramo po končnicah in tipih datotek. Ker želimo omogočiti uporabniku izbiro mape, smo dialog nastavili tako, da prikazuje samo mape in nobenih drugih datotek. Ob potrditvi se shrani izbrana mapa, ob preklicu pa se ohrani stara vrednost. Koda 4.6 prikazuje logiko za izbiranje ciljne mape.

Listing 4.6: Reža za spreminjanje ciljne mape

```
void YoutubeDL::on_browseButton_clicked() {
    QFileDialog dialog(this);
    dialog.setFileMode(QFileDialog::Directory);
    dialog.setOption(QFileDialog::ShowDirsOnly);

    QString path = QFileDialog::getExistingDirectory();
    if (!path.isNull()) {
        ui->path->setText(path);
    }
```

}

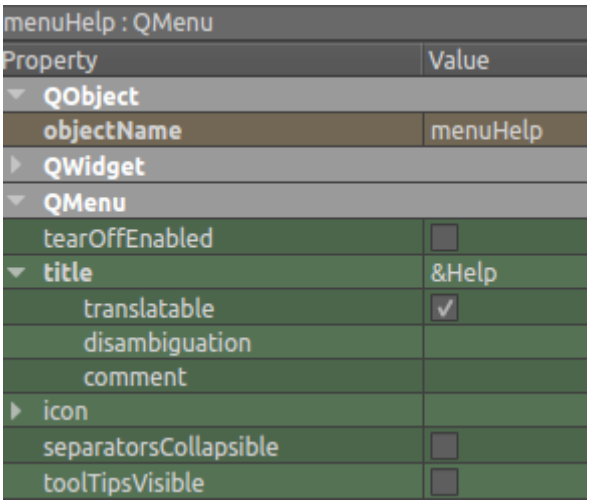
Tretja skupina gradnikov je namenjena prikazu podatkov o spletnem viru. Ko uporabnik poda podprt spletni naslov se pridobijo slika, naslov in opis vira, ki se prikažejo v tem delu.

Naslednja skupina gradnikov je namenjena naprednim nastavitvam. Privzeto je skrita, vendar se s klikom na gumb *Prikaži napredne možnosti* prikaže, tako da uporabnik lahko definira posebne parametre. Vidljivost okvira QFrame se nastavlja z metodo *setVisible(boolean)*.

V zadnji skupini se nahajajo gumb za prenos, indikator stanja prenosa, in gumba za pavzo ter prekinitev. Vsi trije gumbi so ob zagonu aplikacije onemogočeni, in se jih ob določenih pogojih omogoči. Gumb za prenos se omogoči, če se proces pridobivanja informacij uspešno zaključi. Ob kliku na gumb za prenos se prebere naslov, ki ga je podal uporabnik, pot do mape za prenos in morebitni dodatni parametri. V reži, ki se odziva na klik gumba za prenos, se pokliče program za prenos vira. Za prikaz napredka prenosa smo povezali *readyReadStandardOutput* na režo, ki bere standardni izhod procesa od koder dobi informacijo o napredku, ter jo posreduje v indikator stanja prenosa.

4.3.3 Mnemoniki

Mnemoniki omogočajo dostop do akcij v menijih preko tipkovnice. Ponavadi se mnemonik doda na prvo črko imena akcije, do katere lahko potem dostopamo preko tipkovnice s kombinacijo tipk *Alt + <črka>*. Qt omogoča izjemno preprosto dodajanje mnemonikov na elemente v menijih. V atributu *title* objektov tipa *QMenu*, dodamo pred željeno črko znak *&*, in Qt bo poskrbel da se ustvari mnemonik nad črko za znakom. Na sliki 4.2 vidimo konfiguracijo mnemonika v programu Qt Creator.



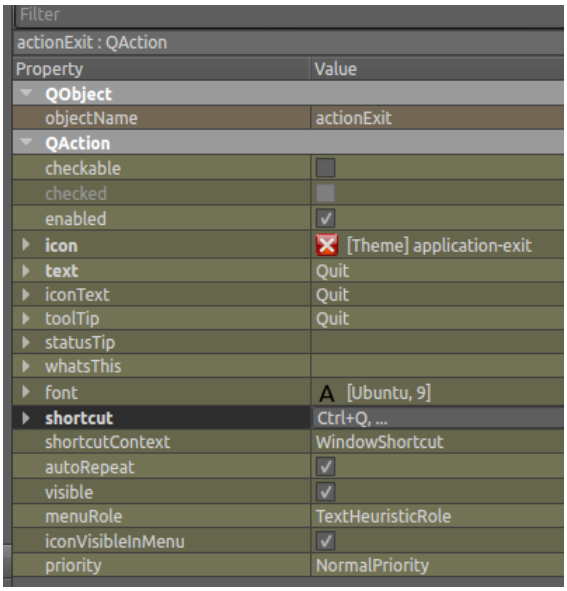
Slika 4.2: Dodajanje mnemonika

4.3.4 Bližnjice

Na elemente tipa *QAction*, ki so po hierarhiji otroci objektov tipa *QMenu*, lahko preprosto dodamo bližnjice za dostop preko tipkovnice. V Qt Creatorju z miško kliknemo na atribut *shortcut* in preko tipkovnice vnesemo željeno bližnjico, kot je vidno na sliki 4.3.

4.3.5 Ikone v meniju

Ikone poleg akcij v meniju lahko preprosto dodamo v Qt Creatorju kot atribut *icon* objekta *QAction*, kjer izbiramo med sistemskimi ikonami in zunanjimi datotekami. Na GNU/Linux distribucijah so ponavadi vse najbolj uporabljane ikone v trenutni temi, ki jo ima uporabnik nastavljeno. Zaradi nestandariziranih direktiv o imenovanju ikon so v preteklosti razvijalci uporabljali različna imena za iste ikone, zaradi česar so ustvarjalci ikon morali podvajati datoteke da bi njihove ikone delovale v različnih okoljih (KDE, Gnome, ...). To je pripeljalo do specifikacije standarda za poimenovanje [11]. Zaradi tega se lahko zanesemo na delovanje ikon s standardnimi imeni na X11 platformi. Na platformah, ki ne podpirajo ikon, vezanih na temo, kot



Slika 4.3: Nastavljanje bližnjice

sta Windows in OS X, lahko podamo lastne ikone, ki se naj prikažejo.

Listing 4.7: Nastavljanje ikone v meniju

```
ui->actionExit->setIcon(
QIcon::fromTheme(QStringLiteral("application-exit"),
QIcon(":images/Actions-application-exit-icon.png")));
```

V zgornjem delu kode je prikazano nastavljanje ikone na akcijo *actionExit*, ki sproži zapiranje programa. Funkcija *setIcon* poskusi najprej nastaviti ikono z imenom *application-exit*, in če funkcija *fromTheme* ne najde ikone, bo poiskala in nastavila ikono ki smo jo predhodno dodali v datoteko z viri (.qrc). S tem smo dosegli da na platformi X11 uporabljamo ikone trenutno nastavljene teme, na ostalih platformah pa kot alternativo uporabimo priložene ikone.

4.3.6 Slike

Za manipulacijo s slikami ima Qt na voljo poseben razred imenovan *QImage*. Če potrebujemo osnovne funkcije kot je npr. prikaz slike, lahko uporabimo

tudi objekte tipa *QLabel*, ki so namenjeni samo prikazovanju besedila in slik. V naš program smo hoteli dodati animirano sliko formata *gif*, ki bi uporabnika obvestila da se izvaja operacija pridobivanja informacij o viru. V ta namen smo v objekt tipa *QMovie*, ki je namenjen prikazovanju animacij brez zvoka, prebrali sliko in jo prikazali v objektu z imenom *titleDescLabel* tipa *QLabel*.

Listing 4.8: Prikaz animirane slike

```
QMovie *movie = new QMovie(":images/loader.gif");
ui->titleDescLabel->setAlignment(Qt::AlignCenter);
ui->titleDescLabel->setMovie(movie);
movie->start();
```

4.4 Ikona

Pomembna stvar pri aplikaciji je tudi ikona, saj uporabnik z njo lažje prepozna program ki ga želi pognati. Zaradi različnih konvencij je bilo potrebno za vsako platformo dodati specifično kodo za dodajanje ikone. Primerno ikono smo našli na spletni strani IconArchive [6], kjer so na voljo različne ikone za komercialno in nekomercialno rabo.

4.4.1 Windows

Za platformo Windows potrebujemo ikono formata *ICO*. Ikono shranimo v vrhnji direktorij našega projekta in dodamo v projektno datoteko naslednjo vrstico:

```
RC_ICONS = YoutubeDL.ico
```

4.4.2 OS X

Za sisteme OS X potrebujemo ikone formata *icns*, in je priporočljivo da jih ustvarjamo s programom *Icon Composer*, ki je del paketa *Apple Developer Tools*. V projektno datoteko dodamo vrstico:

```
ICON = YoutubeDL.icns
```

4.4.3 X11

Zaradi raznolikosti GNU/Linux platform ni enotnega načina za nastavljanje ikone programu. Ponavadi skrbniki programov na repozitorijih vsake distribucije poskrbijo za pravilno pakiranje aplikacije v strukturo, specifično za vsako distribucijo. Zaradi tega bomo ta postopek izpustili.

4.5 Večjezičnost

Qt nam omogoča preprosto internacionalizacijo in lokalizacijo aplikacij. To pomeni, da lahko aplikacijo prilagajamo različnim jezikom brez spreminjanja obstoječe logike in dodajamo komponente specifične za jezike kot so datum in čas, ter prevajamo tekst. Besedilo naše aplikacije je privzeto v angleščini, za platforme ki imajo nastavljeno slovenščino, bomo vse nize, ki jih uporabnik vidi, prevedli in nastavili aplikacijo tako, da ob zagonu preveri jezik sistema in temu primerno prilagodi jezik aplikacije.
Najprej moramo v projektno datoteko dodati vrstico

```
TRANSLATIONS = youtubedl_sl_SI.ts
```

Qt se pri poimenovanju prevodov drži standarda <jezik>_<država>, kjer je jezik dvo oz. tročrkovna oznaka jezika po standardu ISO 639 in država dvočrkovna oznaka po standardu ISO 3166. V ukazni vrstici se premaknemo v direktorij kjer se nahaja naš projekt in izvedemo ukaz

```
lupdate YoutubeDL.pro
```

Ukaz *lupdate* poišče vse nize ki so vidni na uporabniškem vmesniku in ustvari XML datoteko s prevodi *youtubedl_ sl_ SI.ts*, definirano po DTD (ang. document type definition) ki je dostopen na `http://qt-project.org/doc/qt-5/linguist-ts-file-format.html`. Če niza nismo ustvarili kot lastnost elementa v načinu Design, ampak smo ga določili v izvorni kodi, lupdate takšnega niza ne bo zaznal. Zato ga moramo oviti s funkcijo *tr(QString)*.

Listing 4.9: Uporaba funkcije tr

```
QMessageBox::about(this, tr("Supported sites"), videoInfo);
```

Lupdate analizira datoteko .ui in poišče vse XML elemente *string*, ki so gnezdeni v elemente *property* z atributom *name*, ki ima vrednost *text*.

Listing 4.10: Definicija akcije Prilepi v datoteki .ui

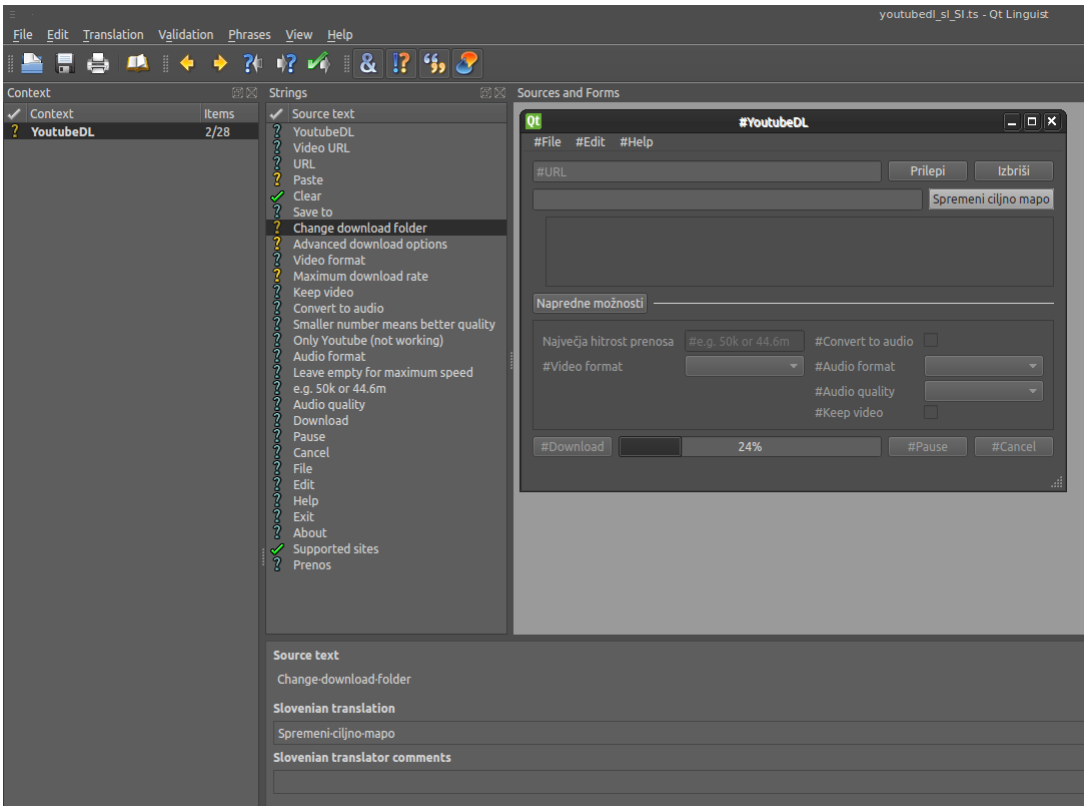
```
<action name="actionPaste">
  <property name="text">
    <string>Paste</string>
  </property>
  <property name="shortcut">
    <string>Ctrl+V</string>
  </property>
</action>
```

Izpis 4.11 prikazuje datoteko, kot jo generira lupdate.

Listing 4.11: Primer datoteke s prevodi

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE TS>
<TS version="2.1" language="sl_SI">
<context>
    <name>YoutubeDL</name>
    <message>
        <location filename="youtubedl.ui" line="55"/>
        <source>Paste</source>
        <translation>Prilepi</translation>
    </message>
</context>
</TS>
```

Po vsakem dodajanju novega niza je potrebno ponovno pognati lupdate, da se spremembe uveljavijo na datoteki XML.
Ustvarjeno datoteko s prevodi odpremo s programom *Qt Linguist*, prikazan na sliki 4.4, ki nam omogoča urejanje datoteke s preprostim grafičnim vmesnikom.



Slika 4.4: Qt Linguist

Qt Linguist samodejno prepozna iz oznak za kateri jezik gre, tako da je treba le še podati prevode za vse nize ki jih želimo prevesti. Če se zgodi, da je prevod enak originalu, preprosto pustimo polje prazno in prikazan bo originalni niz. Ko končamo s prevajanjem je kot zadnji korak potrebno v direktoriju kjer se nahaja projekt pognati ukaz

```
lrelease YoutubeDL.pro
```

Lrelease pretvori .ts datoteke v binarne datoteke formata QM (Qt Message) ki jih uporablja končna aplikacija za prevode. Rezultat je datoteka *youtubedl_sl_SI.qm*. Pri distribuciji aplikacije moramo biti pozorni na to, da se datoteke QM nahajajo v istem direktoriju kot aplikacija oz. jih moramo dodati v datoteko *qrc*, če želimo da se prevodi upoštevajo.

Aplikacija mora ob zagonu preveriti jezik sistema ter poiskati ustrezne prevode in jih naložiti. V ta namen dodamo v funkcijo *main* naslednjo kodo

Listing 4.12: Nalaganje prevodov

```
QApplication a(argc, argv);
QTranslator youtubedlTranslator;
youtubedlTranslator.load("youtubedl_" +
QLocale::system().name());
a.installTranslator(&youtubedlTranslator);
```

Po prevajanju aplikacije lahko preverimo delovanje prevodov na UNIXu podobnih sistemih (GNU/Linux in OS X) tako, da v trenutni seji v terminalu spremenimo spremenljivko LANG. Glede na specifike različnih sistemov bo eden od spodnjih ukazov deloval.

```
export LANG=sl
setenv LANG sl
```

Na platformi Windows v ukazni vrstici poženemo ukaz

```
set LANG=sl
```

Ko je spremenljivka nastavljena, poženemo program iz ukazne vrstice, da preverimo prevode. Če imamo več prevodov, preprosto zamenjamo parameter *sl* v ustrezno oznako za vsak jezik. Pomembno je to, da aplikacijo poženemo iz ukazne vrstice, saj bo sicer program preveril sistemsko nastavitev za jezik.

Podrobnosti o razredih, ki omogočajo prevode in omejitvah za posamezne platforme, so dostopne na qt-project.org/doc/qt-5/internationalization.html.

Poglavje 5

Priprava okolja za prevajanje

V tem poglavju bomo opisali pripravo okolja za prevajanje izvorne kode za vse tri ciljne platforme.

5.1 X11

Za potrebe prevajanja in testiranja smo uporabili GNU/Linux distribucijo Lubuntu [2]. Najprej moramo prenesti in namestiti Qt SDK (Software Development Kit). Dobimo ga na uradni spletni strani projekta Qt na povezavi http://qt-project.org/downloads. Kliknemo na povezavo *Qt <verzija> for Linux 32-bit (<velikost> MB)*. Če delamo na 64 bitnem operacijske sistemu, je priporočljivo izbrati 64 bitno inačico Qt. Prenesena datoteka bo imela ime *qt-opensource-linux-x86-<verzija>.run*. Trenutna najnovejša verzija je 5.3.1. Ko se prenašanje dokonča, je potrebno datoteki dodeliti pravice za izvrševanje. To storimo v grafičnem vmesniku tako, da najprej z desnim klikom na datoteko izberemo *Lastnosti* in potem v zaviku *Pravice* obkljukamo polje za izvrševanje. Ko to storimo je potrebno zagnati datoteko in slediti vodiču za namestitev.
Da nam ne bo potrebno za prevajanje vpisovati absolutne poti do programa *qmake*, si bomo delo olajšali z naslednjim ukazom. V terminal vpišemo:

```
sudo ln /home/darko/Qt5.3.1/5.3/gcc/bin/qmake /usr/bin/qmake5
```

Tako bomo iz terminala lahko poganjali program *qmake* ne glede na to, v katerem direktoriju se nahajamo. Za prevajanje izvorne kode potrebujemo še nekatere programe, ki jih namestimo z ukazom

```
sudo apt-get install build-essential libglu1-mesa-dev
```

V terminalu se prestavimo v direktorij, kjer se nahaja projekt in izvršimo naslednje ukaze:

```
make distclean
qmake5
make
```

Prvi ukaz počisti direktorij za morebitnimi datotekami, ki niso potrebne za prevajanje, ponavadi so to ostanki od prejšnjega prevajanja. Če se prevajanje konča uspešno, se ustvari datoteka ki jo lahko uporabljamo. Poženemo jo preko grafične lupine ali pa iz terminala z ukazom

```
./YoutubeDL
```

5.1.1 Prevajanje s statičnimi knjižnicami

Z uradne strani prenesemo izvorno kodo ogrodja Qt v kompresiranem formatu *tar.gz*, arhiv razširimo in se z ukazno vrstico premaknemo v novonastali direktorij. Poženemo naslednja ukaza:

```
./configure -nomake examples -nomake tests -skip qtwebkit
-opensource -confirm-license -static -release -qt-xcb


make -j3
```

5.2 Windows 8

Postopek je bil izveden na 32 bitnem sistemu Windows 8. Z uradne strani projekta Qt smo najprej prenesli najnovejšo namestitveno datoteko in po

navodilih namestili ogrodje. Pri namestitvi smo pozorni na to, da izberemo tudi namestitev programske zbirke MinGW [4] v primeru, da je še nismo imeli nameščene. Po uspešni namestitvi dodamo v sistemsko spremenljivko *Path* poti do programov *qmake* in *mingw32-make*, ki se nahajata v direktorijih

```
C:\Qt\5.3\mingw482_32\bin
C:\Qt\Tools\mingw482_32\bin
```

V ukazni vrstici se premaknemo v direktorij, kjer se nahaja naš projekt in poženemo naslednje ukaze:

```
mingw32-make distclean
qmake
mingw32-make
windeployqt release
```

Zadnji ukaz zgradi imeniško strukturo z vsemi knjižnicami v direktoriju *release*, ki jih naš program potrebuje za delovanje, s čimer postane distribucija programa izjemno preprosta.

5.2.1 Prevajanje s statičnimi knjižnicami

Z uradne strani prenesemo izvorno kodo ogrodja Qt v kompresiranem formatu *zip*, arhiv razširimo in se z ukazno vrstico premaknemo v novonastali direktorij. Poženemo naslednja ukaza:

```
configure -nomake examples -nomake tests -skip qtwebkit
-opensource -confirm-license -static -opengl desktop -release
```

```
mingw32-make -j3
```

Ko se ukaza izvršita, lahko našo aplikacijo prevedemo statično, z enakim zaporedjem ukazov kot smo prevedli dinamično, le da moramo biti pozorni da spremenimo pot do programa *qmake*, ki je zdaj preveden statično.

5.3 OS X

Najprej je potrebno s programom App Store naložiti brezplačen paket Xcode [3]. Za namestitev potrebujemo brezplačen račun *Apple ID*. Xcode je integrirano razvojno okolje, ki vsebuje zbirko orodij za razvoj programske opreme za sisteme OS X in iOS. Ko se namestitev uspešno dokonča, je potrebno prenesti namestitveno datoteko za Qt z uradne strani. Ime povezave je formata *Qt <verzija> for Mac (<velikost> MB)*. Ime prenesene datoteke bo *qt-opensource-mac-x64-clang-<verzij>.dmg*. Ko se prenos dokonča, zaženemo datoteko in dokončamo namestitev. V primeru da Xcode predhodno ni bil nameščen, bomo primerno opozorjeni. S terminalom se premaknemo v direktorij, kjer se nahaja naš projekt in izvršimo naslednje ukaze:

```
export PATH=$PATH:/Users/darko/Qt5.3.1./5.3/clang_64/bin/
make distclean
qmake
make
macdeployqt YoutubeDL.app -dmg
```

Prvi ukaz nam doda v okoljsko spremenljivko *PATH* pot do programov, ki smo jih v prejšnjem koraku namestili. Ukaz *macdeployqt* ustvari datoteko tipa *Apple Disk Image* s končnico *dmg*, specifičnega za platformo OS X. Ta omogoča preprosto distribucijo prevedenega programa.

Poglavje 6

Preizkus delovanja

Za delovanje vseh funkcionalnosti potrebujemo naslednje programe:

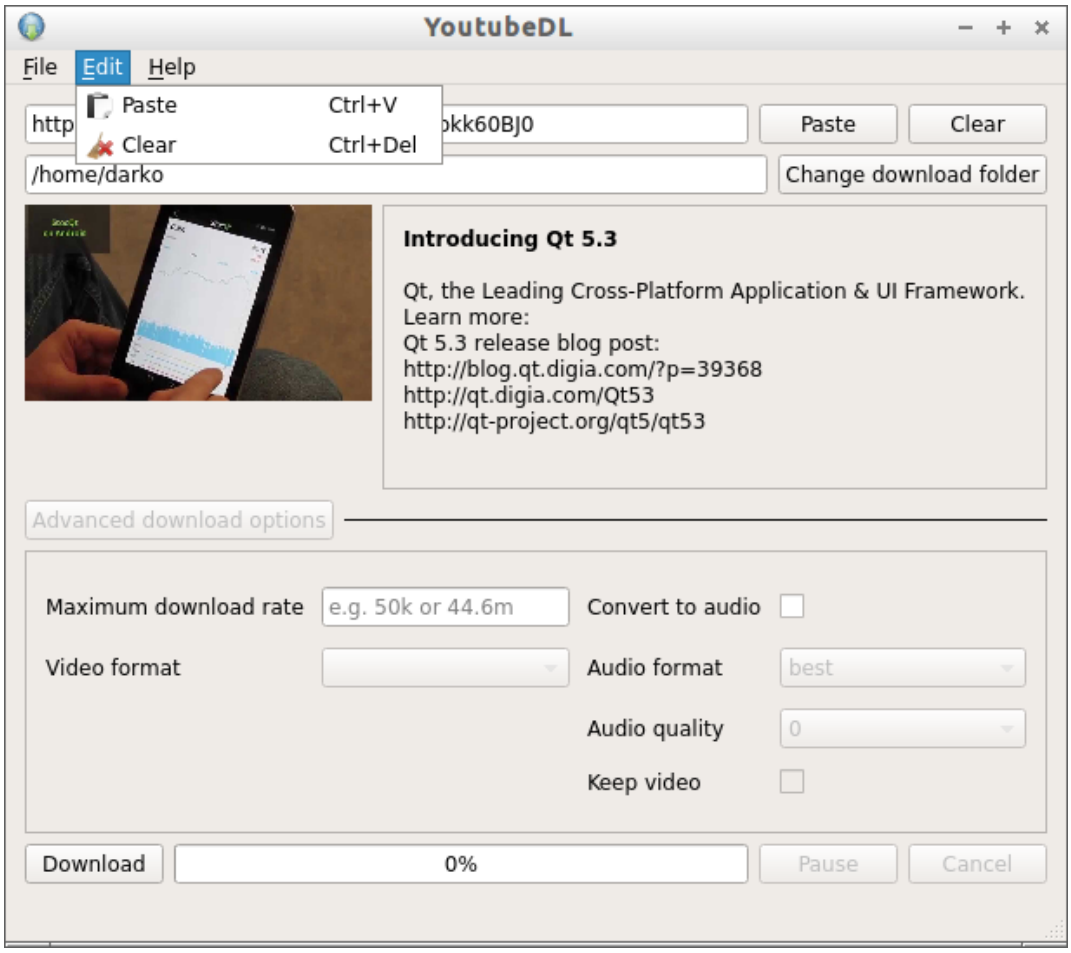
- python
- youtube-dl
- ffmpeg ali avconv (opcijsko)

Programski jezik *python* je potreben za izvajanje programa *youtube-dl*, za namene pretvarjanja med multimedijskimi formati pa je potreben še paket *ffmpeg* ali *avconv*.

6.1 X11

Za namestitev potrebnih programov v terminal vpišemo naslednji ukaz:

```
sudo apt-get install python youtube-dl libav-tools
```



Slika 6.1: Aplikacija na platformi X11

6.2 OS X

Naslednji postopek predvideva da je *python* že nameščen. V terminal vpišemo naslednja ukaza:

```
sudo curl https://yt-dl.org/downloads/latest/youtube-dl -o
/usr/bin/youtube-dl
```

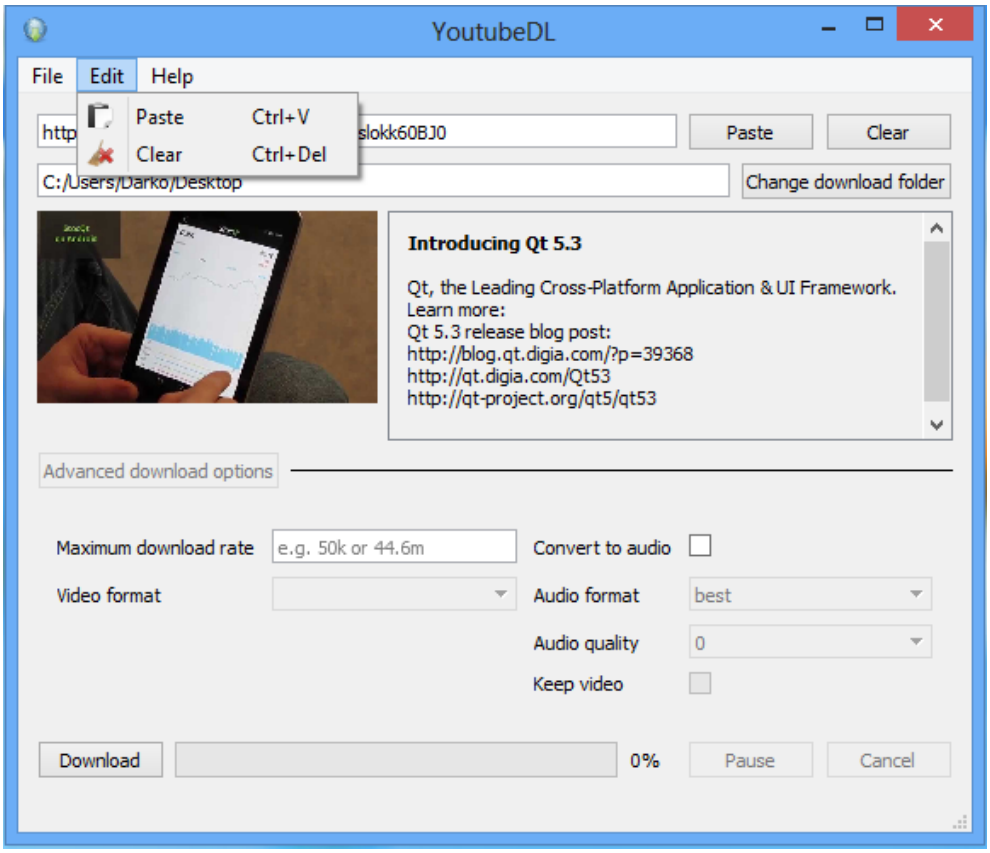
```
sudo chmod a+x /usr/bin/youtube-dl
```

Prvi ukaz prenese in namesti najnovejšo verzijo programa *youtube-dl*, drugi pa omogoči izvrševanje.

Slika 6.2: Aplikacija na platformi OS X

6.3 Windows

Na sistem je potrebno najprej namestiti *python*, nato pa z naslova https://yt-dl.org/downloads/latest/youtube-dl.exe prenesemo najnovejšo različico programa *youtube-dl*. V spremenljivko *PATH* dodamo pot do direktorija kjer se nahaja preneseni program.



Slika 6.3: Aplikacija na platformi Windows

Poglavje 7

Sklepne ugotovitve

V diplomski nalogi smo šli skozi razvoj preproste prenosljive aplikacije z ogrodjem Qt. Najprej smo opisali problem prenosljivih aplikacij, in nato po korakih razvijali našo aplikacijo, ob tem pa podrobneje predstavljali ogrodje Qt. Uspešno smo razvili aplikacijo, ki deluje na platformah Windows 8, OS X in X11 na GNU/Linux. Opisali smo pripravo okolja za prevajanje izvorne kode in pripravili navodila za distribucijo prevedenega programa. Diplomska naloga lahko praktično služi kot vodič z navodili in referencami za razvoj prenosljive aplikacije, hkrati pa je razvita aplikacija splošno uporaben produkt, zaradi javno objavljene izvorne kode pa lahko služi kot ogrodje za razvoj naprednejše aplikacije. Možne izboljšave na aplikaciji so boljši uporabniški vmesnik (razporeditev elementov), vgrajen posodobilnik, ter dodatni vmesnik za pridobivanje informacij o večih virih hkrati. Odzivi prvih uporabnikov so razen prejšnjih predlogov za izboljšave pozitivni in na splošno je aplikacija izbrana kot boljša alternativa podobnim programom.

Literatura

[1] Uradna spletna stran projekta Qt, dostopno na:
http://qt-project.org/

[2] Uradna spletna stran GNU/Linux distribucije Lubuntu, dostopno na:
http://lubuntu.net/

[3] Xcode, dostopno na:
https://developer.apple.com/xcode/

[4] Minimalist GNU for Windows (MinGW), dostopno na:
http://www.mingw.org/

[5] Predavanje *Getting Started Debugging on Linux*, dostopno na:
https://www.youtube.com/watch?v=xTmAknUbpB0

[6] IconArchive, dostopno na:
http://www.iconarchive.com

[7] Razhroščevalniki v ogrodju Qt, dostopno na:
http://qt-project.org/doc/qtcreator-3.0/creator-debugger-engines.html

[8] Manj splošno dovoljenje GNU, dostopno na:
http://qt-project.org/doc/qt-5/lgpl.html

[9] Manj splošno dovoljenje GNU, inačica 2.1, dostopno na:
https://www.gnu.org/licenses/old-licenses/lgpl-2.1.html

[10] Splošno dovoljenje GNU, dostopno na:
https://www.gnu.org/licenses/gpl-3.0.html

[11] Standard poimenovanja ikon na platformi X11, dostopno na:
http://standards.freedesktop.org/icon-naming-spec/icon-naming-spec-latest.html

[12] Vzorec opazovalca, dostopno na:
https://en.wikipedia.org/wiki/Observer_pattern

[13] Signali in reže v ogrodju Qt, dostopno na:
http://qt-project.org/doc/qt-5/signalsandslots.html

[14] Association for Computing Machinery, dostopno na:
http://www.acm.org/

[15] Zbirka prevajalnikov GNU, dostopno na:
https://gcc.gnu.org/

[16] Uradna stran jedra Linux, dostopno na:
https://www.kernel.org/

[17] Referenca orodja nmake, dostopno na:
http://msdn.microsoft.com/en-us/library/dd9y37ha.aspx

[18] Jezik SQL, dostopno na:
https://en.wikipedia.org/wiki/SQL

[19] Orodje make, dostopno na:
https://en.wikipedia.org/wiki/Make_%28software%29

[20] Programski jezik Python:
https://www.python.org/

[21] Program youtube-dl:
https://rg3.github.io/youtube-dl/